Haskell-RL

An Equational Specification of Haskell in Maude

Andrew Bennett

Presented on 24 April 2006

Summary

- Haskell / Haskell-RL Examples
- Supported Features
- Haskell-RL State
- Implementation
  - Functions
  - Data Types
  - Pattern Matching
- Results

Examples
(in Haskell)

Factorial

```
f :: Int -> Int
f 0 = 1
f x = x * f (x - 1)
```

Fibonacci

```
fib :: Int -> Int
fib n
  | n == 0 = 0
  | n == 1 = 1
  | n > 1  = fib(n - 2) + fib(n - 1)
```

List Building

```
f :: Int -> [Int]
f 0 = []
f x = x : (f (x - 1))
```

Data Types

```
data Node = Tree Node Node | Leaf Int
f :: Node -> Int
f (Leaf x) = x
f (Tree a b) = f a + f b
```

Examples
(in Haskell-RL)

Fibonacci

```
f 0 = 1 ;
f x = *(x,f (+((-1),x)))
```

Fibonacci

```
'fib n
  |=(==(n,0),0)
  |=(==(n,1),1)
  |=(>(n,1), (+ ('fib (+(n,-2)),
                ('fib (+(n,-1))))))
```

List Building

```
data 'List = dt('Nil, ())
 || dt('Cons, (Int,'List)) ;
f 0 = dt('Nil, ()) ;
f x = dt('Cons,(x,(f (+(-1,x)))))
```

Data Types

```
data 'Node = dt('Leaf, Int)
 || dt('Tree, ('Node, 'Node)) ;
f dt('Leaf, x) = x ;
f dt('Tree, (x,y)) = +(f x, f y)
```

Examples
(in Haskell-RL)

Permutations

```
'len dt('Nil, $) = 0 ;
'len dt('Cons, ($,'ls)) = (+(1,'len 'ls)) ;
'null dt('Nil, $) = True ;
'null $ = False ;
'perm 0 = dt('Cons, (dt('Nil, ()), dt('Nil, ()))) ;
'perm n = 'insert n ('perm (+((-1),n))) ;
'insert ($ l)  |=('null l, dt('Nil, ())) ;
'insert (n dt('Cons, (l,'ls))) = 'append ('interleave n l) ('insert n 'ls) ;
'interleave (n l)
   |=('null l, dt('Cons, (dt('Cons, (n,dt('Nil, ()))), dt('Nil, ())))) ;
'interleave (n dt('Cons, (l,'ls))) =
       dt('Cons, (dt('Cons, (n,dt('Cons, (l,'ls)))),
                  ('mycons l ('interleave n 'ls)))) ;
'mycons (x l)  |=('null l, dt('Nil, ())) ;
'mycons (x dt('Cons, (l,'ls))) =
       dt('Cons, (dt('Cons, (x,l)),('mycons x 'ls))) ;
'append (u v)  |=('null u, v) ;
'append (dt('Cons, (u,'us)) v) = dt('Cons, (u,('append 'us v)))
```

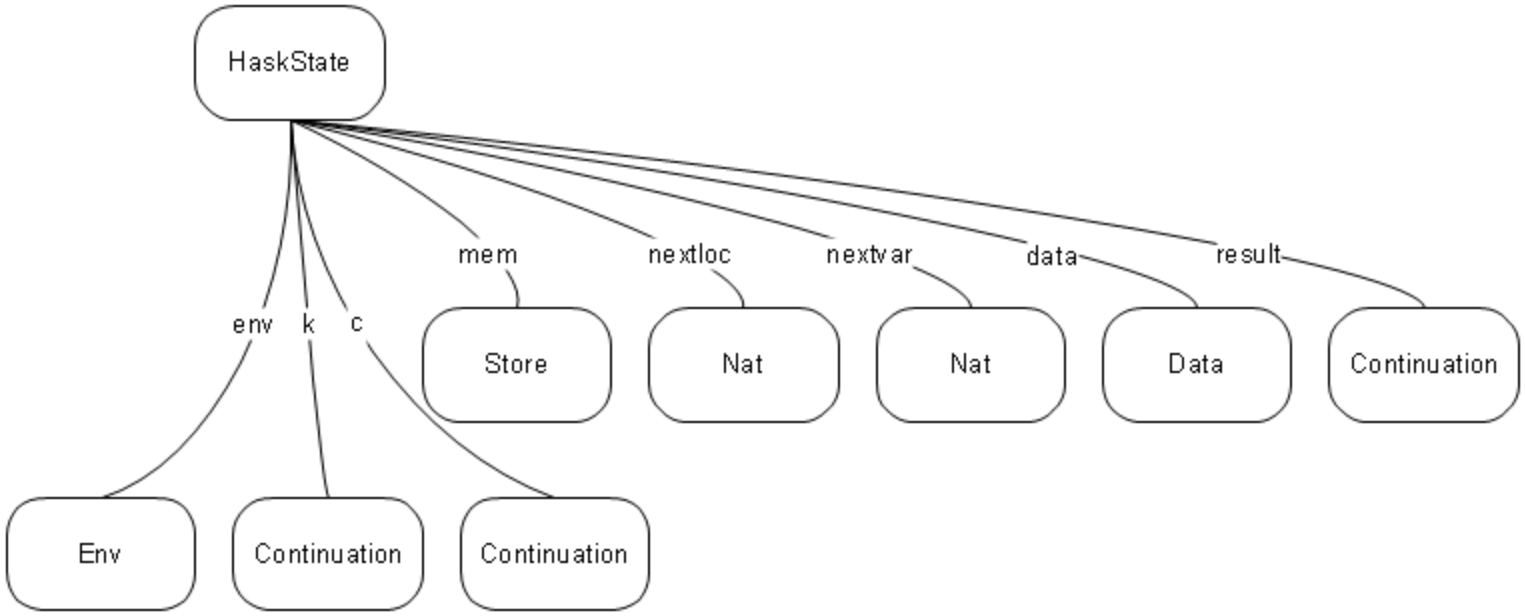
Supported Features in Haskell-RL

- Fully Lazy
- General and Arithmetic Expressions
- Functions
  - Currying, partial application, lambda
- User Data Types
  - Lists, Trees
  - Tuples supported natively
- Simple and Recursive Pattern Matching

Big Features Not Implemented

- “Pretty” Lists, List Comprehensions
- Basic types other than Int (char, float)
- Type Checking
  - All static, actual semantics would not change
- I/O
- Monads
- Classes / Modules

Haskell-RL State Infrastructure



Free Variables

```
f (0 a) = a ;
f (7 $) = 9 ;
f (a 3)
  |=(>(a,3),a) ;
f (a $) = 0
```

- Variable a occurs in multiple binding locations, can’t use it for binding
- Use nextvar state attribute to generate fresh variable (fv(4), etc)

Functions

- Previous slide is translated to:

```
f = \ fv(0) -> \ fv(1) ->
        case Tuple(fv(0),fv(1)) of
         ( Tuple(0,a) -> a ;
           Tuple(7,$) -> 9 ;
           Tuple(a,3) -> if(>(a, 3), a, nomatch) ;
           Tuple(a,$) -> 0 )
```

- Free variables translated through case statement and pattern matching

Data Types

```
data ‘Node = dt(‘Tree, (‘Node, ‘Node)) || dt(‘Leaf, Int)
```

- Ugly syntax
- Without typing, don’t need to specify names in the list
- To use a data type in an expression, surround with same operator
  - `eg dt(‘Tree, (dt(‘Leaf, 3), dt(‘Leaf, 4))`
- Tuples built similarly
  - `eg Tuple(2, (\ x -> x), 4)`

Pattern Matching

- Traverse case one by one
  - Recover environment each iteration
  - Failure => val(bottom)

```
eq k(checkcase(E) -> matchbind($) -> K) = k(good -> K) .
eq k(checkcase(X) -> matchbind(X') -> K) env(Env) =
   k(good -> K) env(Env[X' <- Env[X]]) .
eq k(checkcase(E) -> matchbind(X) -> K) env(Env)
    mem(Mem) nextloc(N) =
   k(good -> K) env(Env[X <- loc(N)]) nextloc(N + 1)
    mem(Mem[loc(N) <- frozen(E, Env, loc(N))]) .
eq k(checkcase(E) -> K) env(Env) =
   k(exp(E) -> K) env(Env) [owise] .
```

Pattern Matching

```
 eq k(val(()) -> matchbind(()) -> K) = k(good -> K) .
 eq k(val(int(I)) -> matchbind(I) -> K) = k(good -> K) .
ceq k(val(int(I)) -> matchbind(I') -> K) = k(casebad -> K)
 if I =/= I' .
 eq k(val(construct(X,Ll)) -> matchbind(dt(X,ECl)) -> K) =
    k(makeande(Ll, ECl) -> K) .
ceq k(val(construct(X,Ll)) -> matchbind(dt(X',ECl)) -> K) =
    k(casebad -> K)
  if X =/= X'

eq k(makeande(L, $) -> K) = k(good -> K) .
eq k(makeande((Ll,L), (ECl, $)) -> K) = k(makeande(Ll, ECl) -> K) .
eq k(makeande(L, E) -> K) = k(checkcasel(L) -> matchbind(E) -> K) .
eq k(makeande((Ll, L), (ECl, E)) -> K) = k(makeande(Ll, ECl) ->
   cand -> checkcasel(L) -> matchbind(E) -> K) .
```

Results

```
reduce in HASK-SEMANTICS : eval(f 0 = 1 ; f x = *(x,f +(-1,x)), f 25000) .
rewrites: 2100071 in 20585ms cpu (20605ms real) (102018 rewrites/second)

reduce in HASK-SEMANTICS : eval(data 'List = dt('Cons, T,T) || dt('Nil, ()) ; 'len
    dt('Nil, $) = 0 ; 'len dt('Cons, $,'ls) = +(1,'len 'ls) ; 'null dt('Nil, $) = True
    ; 'null $ = False ; 'perm 0 = dt('Cons, dt('Nil, ()),dt('Nil, ())) ; 'perm n =
    'insert n ('perm +(-1,n)) ; 'insert ($ l) |=('null l, dt('Nil, ())) ; 'insert (n
    dt('Cons, l,'ls)) = 'append ('interleave n l) ('insert n 'ls) ; 'interleave (n l)
    |=('null l, dt('Cons, dt('Cons, n,dt('Nil, ())),dt('Nil, ()))) ; 'interleave (n
    dt('Cons, l,'ls)) = dt('Cons, dt('Cons, n,dt('Cons, l,'ls)),'mycons l ('interleave
    n 'ls)) ; 'mycons (x l) |=('null l, dt('Nil, ())) ; 'mycons (x dt('Cons, l,'ls)) =
    dt('Cons, dt('Cons, x,l),'mycons x 'ls) ; 'append (u v) |=('null u, v) ; 'append
    (dt('Cons, u,'us) v) = dt('Cons, u,'append 'us v), 'len ('perm 7)) .
rewrites: 6311633 in 62739ms cpu (62771ms real) (100599 rewrites/second)

reduce in HASK-SEMANTICS : eval**(data 'List = dt('Cons, T,T) || dt('Nil, ()) ; 'len
    dt('Nil, $) = 0 ; 'len dt('Cons, $,'ls) = +(1,'len 'ls) ; 'null dt('Nil, $) = True
    ; 'null $ = False ; 'perm 0 = dt('Cons, dt('Nil, ()),dt('Nil, ())) ; 'perm n =
    'insert n ('perm +(-1,n)) ; 'insert ($ l) |=('null l, dt('Nil, ())) ; 'insert (n
    dt('Cons, l,'ls)) = 'append ('interleave n l) ('insert n 'ls) ; 'interleave (n l)
    |=('null l, dt('Cons, dt('Cons, n,dt('Nil, ())),dt('Nil, ()))) ; 'interleave (n
    dt('Cons, l,'ls)) = dt('Cons, dt('Cons, n,dt('Cons, l,'ls)),'mycons l ('interleave
    n 'ls)) ; 'mycons (x l) |=('null l, dt('Nil, ())) ; 'mycons (x dt('Cons, l,'ls)) =
    dt('Cons, dt('Cons, x,l),'mycons x 'ls) ; 'append (u v) |=('null u, v) ; 'append
    (dt('Cons, u,'us) v) = dt('Cons, u,'append 'us v), 'perm 7) .
rewrites: 7801320 in 86581ms cpu (86649ms real) (90103 rewrites/second)
```